ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Realisierung einer Crowd Sensing Anwendung zur Verarbeitung generischer Fragebögen für Windows Phone

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Georg Heinz Eisenhart
georg.eisenhart@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marc Schickler

2016

Fassung 22. Februar 2016

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Durch starke Integration von Smartphones im Alltag ergibt sich die Möglichkeit zu nahezu jedem Zeitpunkt Daten zu erfassen. Diese Daten können nicht nur mit den Sensoren in Smartphones, sondern auch durch direkte Benutzereingaben, zum Beispiel über Fragebögen, erhoben werden. Dies ermöglicht die Entwicklung und den Einsatz von mobilen Anwendungen für verschiedenste Szenarien. Dadurch stellt sich, vor allem für Anwendungen mit Fragebögen, das Problem, dass für jeden Anwendungsfall eine spezielle Lösung entwickelt werden muss.
Fragebögen bestehen zu einem Großteil aus generisch zusammengesetzten Grundelementen, welche in ihrer Struktur immer gleich sind. Somit ist es möglich anhand eines Schemas eine Vielzahl von Anwendungsfällen abzubilden. Im Rahmen dieser Arbeit wird eine mobile Crowd Sensing Anwendung für Windows Phone konzeptioniert und realisiert, welche solche generisch zusammengesetzte Fragebögen, unbeachtet des Anwendungsfalls, verarbeiten kann.

Danksagung

Zuerst möchte ich mich bei Prof. Dr. Manfred Reichert, der mir diese Bachelorarbeit ermöglicht und begutachtet hat, bedanken. Weiterer Dank geht an meinem Betreuer Marc Schickler, der mich während der Bearbeitung unterstützt hat und immer Vertrauen in meine Ideen hatte. Außerdem möchte ich mich bei meiner Familie und meinen Freunden für die mentale Unterstützung und Motivation, sowie fürs Korrekturlesen bedanken.

Inhaltsverzeichnis

1

Einleitung

Die Datenerfassung spielt in der heutigen Gesellschaft eine immer größere Rolle. Durch die starke Integration von Smartphones im alltäglichen Leben ergeben sich immer neue Möglichkeiten großflächig Daten zu erfassen. Messdaten können nicht nur über die im Smartphone eingebauten Sensoren erfasst und übermittelt werden, sondern auch über die direkte Eingabe vom Benutzer. Somit ist auch eine gezielte Abfrage bestimmter Parameter möglich, die in Zusammenhang zu einem spezifischen Kontext stehen.

Dies ermöglicht, auch über längere Zeiträume hinweg, bei einer großen Nutzergruppe Daten zu erfassen. Beispielsweise gibt es im medizinischen Bereich neue Wege zur Diagnose und Überwachung von Krankheiten der Benutzer. Durch die gezielte Abfrage und Auswertung von krankheitsbezogenen Daten ist es möglich nicht nur reine Messwerte, sondern auch umgebungsspezifische Daten im Umfeld des Benutzers zu erfassen, wie zum Beispiel das Wohlbefinden zu verschiedenen Tageszeiten im Alltag des Befragten. Um eine ausführliche Datenerfassung durchzuführen, müssen Anwendungen für Smart-

phones speziell für diesen einen Anwendungsfall entwickelt werden und können somit auch nicht für andere Zwecke eingesetzt werden. Erweiterungen sind meist nur schwer zu realisieren und sehr zeitaufwändig.

1.1 Zielsetzung

Bei der Erfassung solcher Daten können Formulare eingesetzt werden, welche von den Benutzern auf ihren Smartphones bearbeitet werden. Durch eine vordefinierte und einheitliche Strukturierung von Formularen soll es dem Benutzer möglich werden auch verschiedene Fragebögen und Eingabeformulare zu bearbeiten. Hierzu soll eine native Anwendung für Windows Phone entwickelt werden, welche nicht nur ein spezielles bereits fest implementiertes Formular unterstützt, sondern viele heterogene Formulare zur Bearbeitung anbieten kann. Solche Formulare werden zur Laufzeit der Anwendung von einem Server Bezogen. Die Darstellung der Formulare muss dynamisch erzeugt werden, sodass sie vom Benutzer bearbeitet, und die erhobenen Daten an den Server zurückgeschickt werden können.

1.2 Struktur der Arbeit

In Kapitel 2 werden zu Beginn die Grundlagen sowie rudimentäre Aspekte der Anwendung für Windows Phone erklärt und ein kurzer Einblick in verwandte Arbeiten gegeben. Darauffolgend werden in Kapitel 3 die Anforderungen an die Anwendung definiert und erläutert. Der Architektur- und Designentwurf, sowie die Abläufe der Kernfunktionen der Anwendung sind in Kapitel 4 dargestellt. Anschließend wird in Kapitel 5 detailliert auf die technische Umsetzung eingegangen, sowie in Kapitel 6 eine Evaluierung der in Kapitel 3 definierten Anforderungen vorgenommen. Kapitel 7 schließt mit einem Rückblick auf die Arbeit und zeigt zukünftig mögliche Weiterentwicklungen und Erweiterungen der Anwendung.

2

Grundlagen

Im folgenden Kapitel werden die Grundlagen erklärt, sowie die zum Verständnis benötigten Begriffe und die in dieser Arbeit zum Einsatz kommenden Technologien beschrieben.

2.1 Mobile Crowd Sensing

Der Begriff *Crowd Sensing* beschreibt die flächendeckende Datenerfassung, welche zur Datenerhebung durch menschliche Aspekte unterstützt wird. Er setzt sich aus *Crowd* (engl. für *Menschenmasse*) und *Sensing* (engl. für *Erfassen*) zusammen. Mit der wachsenden Anzahl an günstigen und leistungsstarken Sensoren (z.B.: Beschleunigungssensoren, GPS, Mikrophone, Kameras), welche heutzutage in den Mobiltelefonen eingebaut sind, wurde es möglich personen- oder auch gruppenbezogene Daten einfacher zu erfassen. Mussten bisher zur Erhebung von Daten großer Menschengruppen umständlich Befragungen auf der Straße durchgeführt werden oder sogar die Teilnehmer

für eine Studie zu einem bestimmten Ort kommen, kann eine solche Befragung heute zum Beipiel mittels einer Anwendung für Mobiltelefone durchgeführt werden. Durch die einfache Verbreitung solcher Anwendungen über die plattformspezifischen *App Stores* ist es nun möglich mit weniger Aufwand eine größere Anzahl von Teilnehmern miteinzubeziehen und somit auch wesentlich mehr Daten für eine Analyse zu sammeln, als es bisher bei herkömmlichen Umfrageverfahren denkbar war [1].
Hierbei kann die Datenerfassung im Hintergrund ohne eine aktive Teilnahme des Benutzers (*Opportunistic sensing*), oder auch über eine direkte Eingabe von Daten des Benutzers (*Participatory sensing*) erfolgen [2]. Aus diesem Hintergrund heraus bildet sich der Begriff **Mobile Crowd Sensing (MCS)**, welcher in [3] wie folgt formal definiert wird:

> Zitat: *„a new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices, aggregates and fuses the data in the cloud for crowd intelligence extraction and people-centric service delivery“*

2.1.1 Einsatzbereiche von MCS Anwendungen

MCS Anwendungen bieten eine große Bandbreite an Einsatzmöglichkeiten von denen die wichtigsten im Folgenden exemplarisch beschrieben werden.

Sraßenverkehr und Transport

Der Verkehr spielt weltweit eine große Rolle im Alltag - ob Indivdualverkehr, öffentlicher Personennah- und Fernverkehr oder auch im Transportwesen. Um das enorme Verkehrsaufkommen zu bewältigen, können hier MCS Anwendungen helfen den Verkehr zu leiten. Ein Beispiel hierfür ist das *Mobile Millennium*[1] Projekt [4], bei welchem GPS-Daten benutzt werden um Echtzeitinformationen über die Verkehrssituation zu sammeln. Diese werden ausgewertet und an die Benutzer geschickt, um zum Beispiel die voraussichtliche Fahrtdauer anzuzeigen.

[1] http://traffic.berkeley.edu/

Soziale Netzwerke

Für die Erfassung anwendungsspezifischer Daten sind die in Smartphones verbauten Sensoren nicht immer ausreichend. Um die Kommunikationslücke zwischen Smartphones untereinander zu schließen, können soziale Netzwerke sowohl zur Übertragung von Daten, als auch zur Erfassung derselbigen genutzt werden. In der Fallstudie *Crowd-Sourced Weather Radar* wurde die Social Media Plattform *Twitter*[2] benutzt, um mit den von Benutzern eingegeben Daten zur lokalen Wettersituation eine aktuelle Wetterkarte zu generieren. Hierbei wurde eine Art *publish-subscriber* Dienst umgesetzt, bei welchem Wetteranfragen für Orte gestellt werden und die Benutzer mit fest vorgegebenen Werten (z.B.: 0 für sonnig, 1 für bewölkt, etc.) antworten können [5].

Umweltüberwachung und Analyse

Mit MCS Anwendungen ist es, den Möglichkeiten von Smartphones entsprechend, beispielsweise möglich in Ballungsräumen die Benutzer aktiv an der Datenerhebung zur Umweltbelastung zu beteiligen.
Ein solches Szenario ist in der Veröffentlichung *Citizen Noise Pollution Monitoring* [6] dargestellt. Den Bürgern wurde in dieser Studie ermöglicht, sich an Messungen zur Lärmbelastung aktiv zu beteiligen. Hierbei sind nicht nur Daten zum Standort und des Schalldruckpegels gemessen worden, sondern auch zusätzliche Kontextdaten, wie genauere Angaben zur Geräuschquelle (Verkehr, Nachbarn, ...), zur individuellen Wahrnehmung der gemessenen Geräusche und weitere Standortdetails (zu Hause, auf Arbeit, Name der U-Bahnstation, ...). Mit den hiermit gesammelten Daten konnten Lärmbelastungskarten mit *Google Earth* erstellt werden, bei welchen die Benutzer nicht nur das Gesamtergebnis, sondern auch ihre eigenen Messwerte, oder die Ergebnisse nach definierten Begriffen gefiltert betrachten können [6].

[2] https://twitter.com/

Gesundheitswesen

Viele weit verbreitete Krankheiten (z.B.: Tinnitus, Migräne und chronische Schmerzen) sind schwer und meist nur mit individuellen Therapien behandelbar. Um diese herauszufinden müssen über einen größeren Zeitraum Daten zum Krankheitsbild gesammelt werden. Dies bedingt einen langen kosten- und arbeitsintensiven Klinikaufenthalt. In der Fallstudie [7] wurde eine MCS Anwendung (*TrackYourTinnitus*) entwickelt, bei der die Patienten über einen langen Zeitraum zu zufälligen Tageszeiten einen Fragebogen zur Beurteilung des Tinnitus (Wahrnehmung und spezifische Parameter) ausfüllen mussten. Des Weiteren wurde vom Smartphone während der Bearbeitung des Fragebogens die Umgebungslautstärke gemessen. Diese Daten können zur Bestätigung und Verbesserung der Diagnose aus dem klinischen Aufenthalt verwendet werden. [7].

2.1.2 Skalierung von MCS Anwendungen

MCS Anwendungen können wie in Abbildung 2.1 dargestellt in verschiedenen Größenordnungen von individuell bis global agieren. Hierbei unterscheidet man zwischen den folgenden drei Kategorien, wobei die Übergänge fließend sind:

Persönliche MCS Anwendungen (engl. *individual sensing applications*) sind für Nutzung von Einzelpersonen entwickelt und zielen meist auf die individuelle Datenerfassung und Analyse ab. Die durch solche Anwendungen gesammelten Daten werden meist nur für private Auswertungen genutzt und nicht mit anderen geteilt. Eine Ausnahme gibt es zum Beispiel bei MCS Anwendungen im Bereich des Gesundheitswesens. Die erhobenen Daten werden hier oft zur Diagnose oder auch zur Überwachung des Krankheitsverlaufs mit dem behandelnden Arzt geteilt [1].

Gruppenbezogene MCS Anwendungen (engl. *group sensing applications*) erheben Daten zur Veranschaulichung von Ergebnissen und der Entwicklung eines gemeinsamen Ziels oder Problems einer Interessengruppe.

Gemeinschaftsbezogene MCS Anwendungen (engl. *community sensing applications*) sind meist nur bei großen Teilnehmerzahlen, wie zum Beispiel bei der Verkehr-

und Stauanalyse im *Mobile Millennium* Projekt [4] sinnvoll einsetzbar. Solche Anwendungen spiegeln großflächige Datenerhebungen wieder, welche zur Analyse und auch zur Verbesserung des Allgemeinwohls eingesetzt werden können [1].

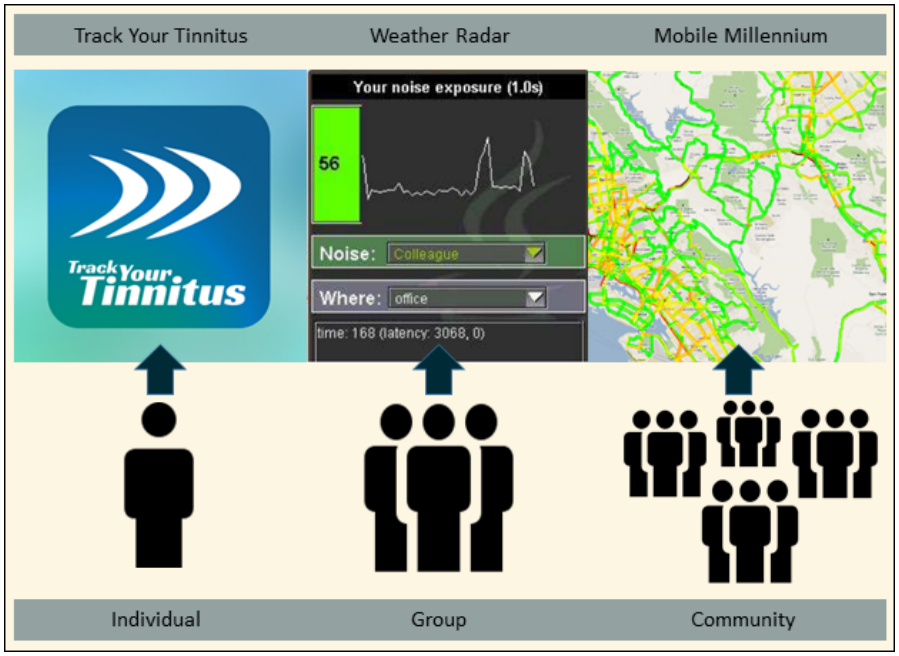

Abbildung 2.1: Skalierung von MCS Anwendungen nach [1]

2.1.3 Probleme bei MCS Anwendungen

Durch die große Anzahl an Möglichkeiten mit dem Smartphone Daten zu sammeln ergeben sich aber auch Probleme unterschiedlichen Ursprungs. Im Folgenden werden die wichtigsten Probleme aufgezeigt, die bei der Entwicklung und dem Einsatz von MCS Anwendungen berücksichtigt werden sollten.
Vor allem *Gruppenbezogene* und *Gemeinschaftsbezogene* MCS Anwendungen benötigen oftmals eine kritische Menge an Benutzern. Um diese zu gewinnen und unter Umständen auch für eine regelmäßige aktive Teilnahme zu motivieren, gibt es verschiedene Ansätze. Für eine Teilnahme können die Benutzer zum Beispiel durch Unterhaltungsaspekte und Informationen ihres Interessengebietes oder auch über Anreize durch Gewinnspiele und Bezahlung motiviert werden.

Bei der Sammlung von sensitiven Daten ist es wichtig das Vertrauen des Benutzer zu haben. Oftmals sind Benutzer nicht bereit private Informationen wie zum Beispiel ihren aktuellen Standort von der Anwendung abrufbar zu machen. Deshalb ist es bei MCS Anwendungen wichtig, verantwortungsvoll mit der Privatsphäre und den Benutzerdaten umzugehen und dem Benutzer eine Möglichkeit zu bieten die Datensammlung zu pausieren.
Ein weiteres Problem stellt die Qualität der erhobenen Daten dar. Diese kann, bedingt durch das Nutzerverhalten, sehr stark schwanken. Hierbei können wechselnde Benutzerzahlen und nicht vorhersehbare Bewegungsmuster in der Öffentlichkeit (z.B.: im Straßenverkehr) die erhobenen Daten beeinflussen und somit das Ergebnis verzerren. Vor allem bei der passiven Datenerfassung über die im Smartphone eingebauten Sensoren weicht die Qualität und Genauigkeit oft voneinander ab. Beispielsweise sind GPS Daten ungenau, wenn sich der Benutzer innerhalb eines Gebäudes befindet, da hier meist der dem System zuletzt bekannte Standpunkt verwendet wird. Ebenso gibt es bei der passiven Datenerfassung von Umgebungsgeräuschen über das Mikrophon markante Unterschiede, je nachdem ob das Smartphone gerade in der Hand gehalten wird oder sich in einer Tasche befindet. Deshalb ist es wichtig zur Qualitätssteigerung die Daten im Voraus zu prüfen und Fehler herauszufiltern [1, 3, 8].

2.2 REST API

Eine REST-API (engl. **Re**presentational **S**tate **T**ransfer) ist eine auf dem *HTTP-Protokoll* basierende Programmierschnittstelle für die Datenübertragung und Darstellung von Programmzuständen. Eine REST API besteht aus Verkettungen von miteinander verbundenen Ressourcen. Wie in Abbildung 2.2 dargestellt, findet die Kommunikation eines sogenannten *RESTful* services direkt mit dem Client statt.

Hierfür werden bei REST-Anfragen URIs auf die Ressourcen abgebildet. So existiert für eine bestimmte Ressource immer die gleiche URI. Die Serveranfragen erfolgen über die *HTTP-Methoden* `GET`, `POST`, `PUT` und `DELETE`. Hierbei werden vom Client die `GET` Methode zum Anfordern, und die `DELETE` Methode zum Löschen von Daten verwendet.

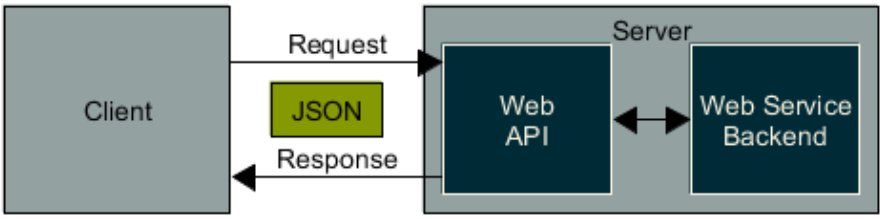

Abbildung 2.2: Aufbau eines RESTful Service nach [9]

`POST` und `PUT` werden verwendet um Daten zum Server zu schicken. Hierbei wird `POST` zum erstellen neuer Ressourcen und `PUT`, da idempotent, zur Überschreibung von Ressourcen verwendet [9, 10].

2.3 Windows Phone

Bei der Entwicklung von Anwendungen für Windows Phone 8.1 gibt es einige Aspekte und Paradigmen die zu beachten sind. Diese und die für die Umsetzung verwendeten Tools werden im Folgenden vorgestellt.

2.3.1 XAML in Windows Phone

XAML (e**X**tensible **A**pplication **M**arkup **L**anguage) ist eine spezielle Form von XML und wird bei der Entwicklung von Anwendungen als wichtiger Teil der *.NET* Plattform als Markup Sprache für das *User Interface* verwendet. Neben der Beschreibung von grafischen Elementen wird XAML auch dazu verwendet die Inhalte, Eigenschaften und Events von Objekten zu setzen. Ein einfaches Beispiel ist in Listing 2.1 dargestellt. Hier wird ein Button Element mit verschiedenen Style Attributen erzeugt. Zusätzlich wird ein *Klickevent* mit dem Namen „ButtonClickEvent“, für welches die Logik in einer separaten Datei (*Code Behind*) implementiert ist, auf das Objekt registriert [11].

```
<Button Content="Click here"
        HorizontalAlignment="Left"
        Margin="10,100,0,0"
        VerticalAlignment="Bottom"
        Click="ButtonClickEvent"
        Background="{StaticResource PhoneAccentBrush}"
        Width="200"
        Height="100"
\>
```

Listing 2.1: Beispiel für ein Button Element in XAML

2.3.2 MVVM und Bindings

Das wichtigste Design Pattern bei der Entwicklung von Windows (Phone) Anwendungen ist das *MVVM* (Model-View-ViewModel) Pattern. Hierbei liegt der Hauptaspekt auf der Kapselung zwischen dem *Model* (Datenmodell), der *View* (Präsentationsschicht der grafischen Oberfläche) und dem *ViewModel*, welches das Bindeglied zwischen Model und der View darstellt und aggregierte Daten aus den Models an die View weiterleitet [11].
Auf diesem Design Pattern stützt sich das Prinzip der *Bindings*. Wie in Abbildung 2.3 dargestellt, werden die Daten aus der Präsentationsschicht (*Binding Target*) mit dem Datenmodell (*Binding Source*) verknüpft. Für Windows Phone gibt es drei verschiedene Modi von Bindings. Beim *OneTime* Binding werden die Daten einmalig beim Erstellen des Bindings aus der Quelle geladen. Der Standardfall ist das *OneWay* Binding, bei dem die Daten beim Erstellen des Bindings geladen und zusätzlich bei jeder Änderung in der Quelle aktualisiert werden. Die letzte Möglichkeit ist das *TwoWay* Binding, bei dem sich Quelle und Ziel immer gegenseitig aktualisieren, wenn beim jeweils anderen Änderungen auftreten [12].

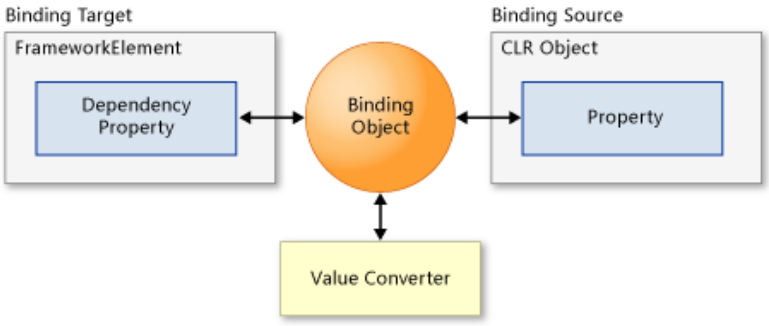

Abbildung 2.3: Grundlegendes Konzept von Bindings [12]

2.3.3 Verwendete Tools

Für das Design und die Umsetzung der Anwendung wurden verschiedene Tools und Programme verwendet. Das **Microsoft Powerpoint Storyboarding** Tool bietet eine Palette an nativen UI-Elementen für Windows und Windows Phone, mit dem die Mockups sowie der klickbarer Prototyp für die Anwendung erstellt wurde. Für die Umsetzung des UI-Entwurfs wurde mit **Microsoft Blend** das fertige User Interface erstellt, welches im Anschluss direkt in das Projekt in Visual Studio importiert werden konnte. Die mobile Anwendung wurde mit **Microsoft Visual Studio** als native Entwicklungsumgebung für *Windows* umgesetzt.

2.3.4 Design Patterns

Neben dem in Kapitel 2.3.2 vorgestellten MVVM Design Pattern werden im Folgenden weitere, bei der Implementierung der Anwendung verwendeten Design Patterns kurz vorgestellt.
Beim **Composite Design Pattern** werden einzelne Komponenten und Gruppen von Komponenten hierarchisch geordnet, sodass sie auf ähnliche Weise verarbeitet werden können. Ein Beispiel hierfür ist die Baumstrukturierung von Elementen zu einer Komponente, welche dann als einzelnes Element behandelt werden und somit wieder in die Baumstruktur eingefügt werden können.

Das **Observer-Pattern** dient zur Festlegung von Beziehungen zwischen Objekten und davon abhängigen Strukturen, welche sich gegenseitig bei Zustandsänderungen informieren können. Dieses Prinzip wird zum Beispiel bei *Bindings* (siehe Kapitel 2.3.2) verwendet, um Änderungen im Benutzerinterface an die zugehörige Datenstruktur weiterzugeben.
Mit dem **Iterator Pattern** wird der Ansatz verfolgt sequenziell auf Elemente von *Collections* zuzugreifen. Hierbei spielt es keine Rolle in welcher Weise die Collection strukturiert ist. Durch die Implementierung des Enumerator Interfaces (in C#: *IEnumerable*) können dann einzelne Elemente oder (unter)Collections zurückgeliefert werden. Auf diesem Pattern basierend, sind auch Features wie *LINQ*[3] (engl. **L**anguage **IN**tegrated **Q**uery) in C# nutzbar [13].

[3] https://msdn.microsoft.com/de-de/library/bb397906.aspx

3

Anforderungen

In diesem Kapitel werden die funktionalen und nichtfunktionalen Anforderungen an die Anwendung definiert und nach ihrer Relevanz bewertet. Eine hohe Priorität spiegelt nicht nur die Anforderung als Kernfeature, sondern auch die Wichtigkeit dieser Anforderung in ihren beschriebenen Aspekten für die Umsetzung wieder. Diese so definierten und bewerteten Anforderungen dienen als Beschreibung des Systemumfangs und als Leitfaden für die Implementierung. In Kapitel 6 werden diese Anforderungen mit dem tatsächlichem Ist Zustand der Anwendung abgeglichen und nach ihrer Umsetzung bewertet.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die Funktionsweise des Systems und wie der Benutzer damit interagieren soll. In Tabelle 3.1 sind die Anforderung mit der entsprechenden Bewertung aufgelistet und werden im Anschluss näher spezifiziert.

Nr.	Anforderung	Priorität
FA01	Beschreibung von Formularen	5
FA02	Versionsunterstützung	4
FA03	Aufgaben vom Server laden	5
FA04	Formulare von Server laden	5
FA05	Anzeige der Formulare für die Bearbeitung	5
FA06	Benutzerauswahl der verfügbaren Aufgaben	4
FA07	Bearbeitung der Fragebögen	5
FA08	Datenerfassung der bearbeiteten Aufgaben	5
FA09	Antwortschema	5
FA10	Zurücksenden der Daten an den Server	5
FA11	Behandlung von Rückgabewerten	3
FA12	Neue beziehungsweise abhängige Aufgaben anzeigen	4
FA13	Grundlegende Benutzereinstellungen und Informationen	2

(5 essentiell, 4 sehr wichtig, 3 wichtig, 2 erwünscht, 1 optional)

Tabelle 3.1: Funktionale Anforderungen

FA01 Beschreibung von Formularen

Damit die Funktionalität der Formulare und der serverseitigen Services gewährleistet bleibt, müssen sich Formulare als Kernbestandteil der Anwendung an die vorgegebenen Spezifikationen halten.

FA02 Versionsunterstützung

Da sich die Beschreibung der Formulare ändern kann, muss die von der Anwendung unterstütze Versionsnummer der Formularbeschreibung an den Server mit übermittelt werden. Des Weiteren muss die Anwendung in der Lage sein die vom Server geschickten Formulare auf ihre Version hin zu prüfen und dann auf die entsprechenden Funktionen zurückgreifen.

FA03 Aufgaben vom Server laden

Beim Starten der Anwendung sollen die für den Benutzer verfügbaren Aufgaben vom Server geladen und in einem Auswahlmenü dargestellt werden.

FA04 Formulare vom Server laden

Bei der Auswahl einer Aufgabe soll das entsprechende Formular vom Server geladen werden. Dies geschieht über Anfragen an einen REST-Service.

FA05 Anzeige der Formulare für die Bearbeitung

Die vom Server übermittelten Daten im JSON Format müssen eingelesen und angezeigt werden, damit der Benutzer die verfügbaren Aufgaben auswählen kann. Sofern durch die Logik vom Server keine Abhängigkeiten zwischen den Aufgaben gegeben sind, werden alle Formulare zur Auswahl angezeigt.

FA06 Benutzerauswahl der verfügbaren Aufgaben

Der Benutzer soll die Bearbeitungsreihenfolge der Aufgaben, sofern dies vom Server nicht anders vorgegeben ist, in beliebiger Reihenfolge auswählen können. Sind Abhängigkeiten von Aufgaben vom Server gegeben, so werden diese automatisch nach dem Beenden der vorherigen Aufgabe dem Benutzer zur Bearbeitung angezeigt.

FA07 Bearbeitung der Fragebögen

Die zur Laufzeit der Anwendung erzeugten Formulare müssen bearbeitbar sein. Dies bedeutet zum Beispiel Textfelder müssen ausfüllbar, Checkboxen auswählbar und Buttons funktionsfähig sein. Hierbei sollen bereits während der Bearbeitung die eingegebenen Daten validiert werden.

FA08 Datenerfassung der bearbeiteten Aufgaben

Nach dem Bearbeiten der Aufgaben durch den Benutzer müssen die Formulare auswertbar sein, damit die erfassten Daten verarbeitet werden können. Hierzu werden die vom Server übermittelten Patterns zur Validierung der jeweiligen Eingabe verwendet.

FA09 Antwortschema

Für die Übermittlung der erfassten Daten müssen diese den vorgegebenen Datentypen des jeweiligen Formularfeldes entsprechen. Diese Daten werden anschließend über ein festgelegtes JSON-Antwortschema an den Server übermittelt.

FA10 Zurücksenden der Daten an den Server

Beim Senden des Antwortschemas an den Server validiert dieser ebenfalls die übermittelten Daten. Hierbei werden vom Server Rückgabewerte zurückgeschickt, welche Auskunft über den Status der Übermittlung und gegebenenfalls detaillierte Informationen zu Fehlern enthalten.

FA11 Behandlung von Rückgabewerten

Die Rückgabewerte des Server, als HTTP-Statuscode, und gegebenenfalls genaueren Fehlerinformationen im JSON-Format, sollen von der Anwendung ausgewertet und behandelt werden können.

FA12 Neue beziehungsweise abhängige Aufgaben anzeigen

Nach dem Bearbeiten eines Formulars sollen, falls verfügbar, weitere Formulare für den Benutzer zur Auswahl stehen oder direkt an ein vom vorherigen Formular abhängiges Folgeformular weitergeleitet werden. Bereits bearbeitete Formulare sollen nicht mehr angezeigt werden.

FA13 Grundlegende Benutzereinstellungen und Informationen

Der Benutzer soll in der Lage sein, Einstellungen zu seinem Profil zu ändern, sowie den Internetzugriff der Anwendung zu regeln (z.B. nur im WLAN verwenden) und darüber entscheiden können, ob die Anwendung Benachrichtigungen erstellen darf. Zusätzlich

werden dem Benutzer Informationen und Statistiken über bereits ausgefüllte Formulare und übermittelte Daten angezeigt.

3.2 Nicht-Funktionale Anforderungen

Die nichtfunktionalen Anforderungen beschreiben die technischen Besonderheiten und Voraussetzungen der Anwendung. In Tabelle 3.2 sind die Anforderung mit der entsprechenden Bewertung aufgelistet und werden im Anschluss genauer spezifiziert.

Nr.	Anforderung	Priorität
NF01	Plattformspezifisches Design	5
NF02	Toleranz gegenüber Fehleingaben	3
NF03	Toleranz gegenüber Umgebungsfehlern	5
NF04	Performanz	3
NF05	Wartbarkeit	5
NF06	Erweiterbarkeit	4
NF07	Absturzfreiheit	4
NF08	Datenschutz	3
NF09	Lokalisierung	2

(5 essentiell, 4 sehr wichtig, 3 wichtig, 2 erwünscht, 1 optional)

Tabelle 3.2: Nichtfunktionale Anforderungen

NF01 Plattformspezifisches Design

Um dem Nutzer eine möglichst geringe mentale Belastung zu bieten, soll das Design und das Bedienkonzept an die Plattform angepasst werden. Hierzu werden die für Windows Phone üblichen *Design Guidelines* verwendet.

NF02 Toleranz gegenüber Fehleingaben

Das System soll zufällige und bewusste Fehleingaben durch den Benutzer erkennen und mit entsprechend informativen Rückmeldungen darauf reagieren.

NF03 Toleranz gegenüber Umgebungsfehlern

Fehler, die aus der Umgebung der Anwendung kommen sollen von der App durch eine Rückmeldung an den Benutzer behandelt werden. Hierbei sind Server-Kommunikationsfehler mit eingeschlossen.

NF04 Performanz

Ladezeiten der Applikation werden durch effiziente Programmierung und Nebenläufigkeit minimal gehalten. Die Anwendung soll auf Interaktionen innerhalb von einer Sekunde reagieren. Werden zeitaufwändigere Methoden ausgeführt wird dem Benutzer eine angemessene Rückmeldung über den Fortschritt des Prozesses geboten.

NF05 Wartbarkeit

Eine strukturierte Implementierung der Anwendung und Dokumentation des Quellcodes soll die zukünftige Wartung bestmöglich gewährleisten.

NF06 Erweiterbarkeit

Die möglichst modular aufgebaute Architektur der Anwendung soll es ermöglichen zusätzliche Erweiterungen einfach zu integrieren.

NF07 Absturzfreiheit

Die Anwendung soll im Normalbetrieb fehlerfrei und absturzfrei ausgeführt werden. Eine stabile Ausführung der Anwendung ist sehr wichtig, da bei der Datenerfassung keine Integritätsprobleme auftreten sollen.

NF08 Datenschutz

Benutzerdaten und Daten, die an den Server übertragen werden, sollen durch eine sichere Authentifizierung und Übertragungsschnittstelle geschützt werden.

NF09 Lokalisierung

Die Anwendung soll nach Möglichkeit auf mehrere Sprachen erweitert werden können.

4

Architektur und Design

Im folgenden Kapitel werden ein Überblick des Architekturentwurfs der Anwendung gegeben sowie der Designentwurf vorgestellt. Des Weiteren werden die grundlegenden Prozesse und Abläufe der Anwendung beschrieben.

4.1 GUI Design

Beim Design der Benutzeroberfläche wurde darauf geachtet, das native Design von Windows Phone beizubehalten, um dem Benutzer eine native Bedienung der Anwendung zu bieten. Im folgenden Abschnitt ist der Entwurf der Benutzeroberfläche, sowie der damit verbundene Dokumentenfluss dargestellt. Abbildung 4.1 zeigt die Interaktionsschritte vom Starten der Anwendung, nachdem der Benutzer eine Benachrichtigung vom Server erhalten hat oder den Start selbst initialisiert hat, über die Startseite der Anwendung, bis

zur Auswahl der für den Benutzer verfügbaren Formulare.

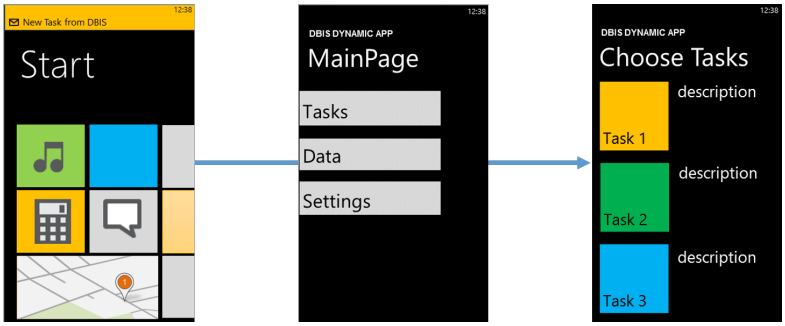

Abbildung 4.1: Dokumentenfluss 1/3

Nachdem der Benutzer im Startmenü den Reiter *Tasks* ausgewählt hat, werden die für den Benutzer verfügbaren Formulare in einer Auswahlliste dargestellt. Die in Abbildung 4.2 veranschaulichten Schritte zeigen wie ein Formular bearbeitet wird und anschließend entweder direkt mit dem nächsten Formular fortgesetzt wird, oder ein anderes Formular selbst ausgewählt werden kann. Nach Beendigung des letzten Formulars wird der Benutzer benachrichtigt und wieder auf die Startseite geleitet.

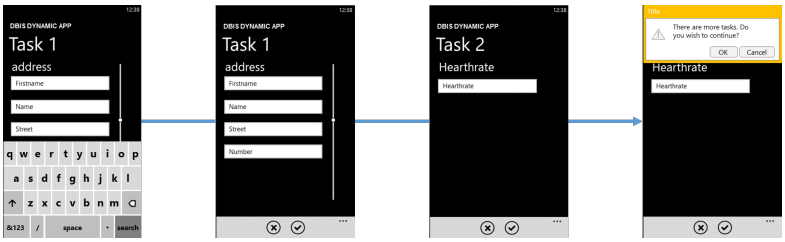

Abbildung 4.2: Dokumentenfluss 2/3

Abbildung 4.3 zeigt weitere Optionen, die für den Benutzer von der Startseite aus verfügbar sind. Unter dem Punkt *Data* können Statistiken und Auswertungen der Formulare angesehen werden und unter *Settings* können grundlegende Einstellung der Anwendung vorgenommen werden.

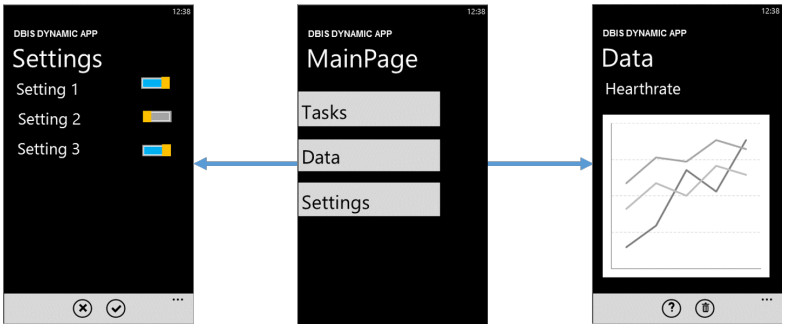

Abbildung 4.3: Dokumentenfluss 3/3

4.2 Entwurf

Bei der Softwarearchitektur wird darauf geachtet, dass die Anwendung möglichst modular strukturiert ist, so dass die Integration von Erweiterungen einfach bleibt. Im Zuge des Entwurfs wird die teilweise oder vollständige Implementierung verschiedener Design-Patterns eingeplant.

4.2.1 Architektur

Nachfolgend sind die Einsatzbereiche für die in Kapitel 2.3.4 vorgestellten Design Patterns aufgelistet.

- **MVVM Pattern** Wird als grundlegendes Design Pattern für die Architektur und Implementierung der Anwendung verwendet.
- **Composite Design Pattern** Kommt bei der Konzeptionierung und Umsetzung des Datenmodells zum Einsatz.
- **Observer Pattern** Dient zur Datensynchronisation zwischen dem Benutzerinterface und der Datenstruktur. Des Weiteren wird dieses Design Pattern bei der Realisierung der Echtzeitüberprüfung bei der Dateneingabe in Formularen angewendet.
- **Iterator Pattern** Wird für die Implementierung für Interfaces verwendet um einfacher auf Objekte in der Datenstruktur zugreifen zu können.

Um zusätzliche Abstraktionsebenen für einen modularen Aufbau zu bieten, wird die Anwendung in verschiedene Packages gegliedert, welche im Folgenden näher erläutert werden. Eine Übersicht der Anwendungsarchitektur ist in Abbildung 4.4 dargestellt.

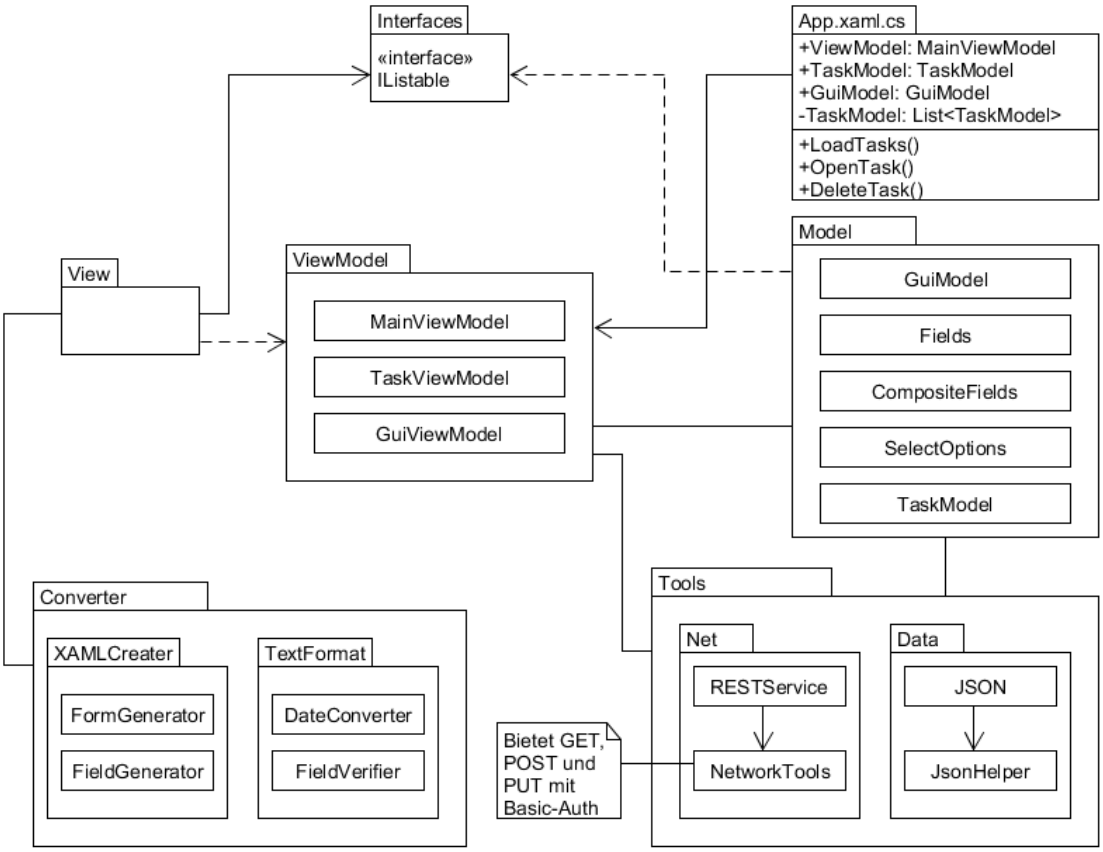

Abbildung 4.4: Architekturübersicht der Anwendung

Das Tools-Package

Das `Tools`-Package besteht aus rein funktionalen Modulen. Hierbei wird die Funktionalität in Daten- und Netzwerkmodule eingeteilt. Die statische Klasse `Networktools` im `Net`-Package bietet die Schnittstelle für GET und POST Requests an den Server, welche von den implementierten Methoden in der `RESTService`-Klasse verwendet werden, um mit dem Server zu kommunizieren.

Das `Data`-Package wird durch die Klassen `JSONHelper`, welche grundlegende Funktionen für den Umgang mit JSON implementiert, und der `JSON`-Klasse gebildet.

Das Converter-Package

Im `Converter`-Package werden hauptsächlich die, für die Anzeige der Formulare benötigten, Interpreter-Funktionen implementiert. Hierfür ist das `XAMLCreater`-Package zuständig, welches aus einer `FormGenerator`- und einer `FieldGenerator`-Klasse besteht. Diese konvertieren das vom Server empfangene JSON-Format in das für WindowsPhone 8.1 anzeigbare XAML-Format. Für die spätere Überprüfung der, vom Benutzer, eingegeben Formulardaten ist die Klasse `TextFormat` zuständig.

Das TaskViewModel

Da in der Anwendung für den Benutzer auch mehrere Aufgaben zur Verfügung stehen können, besitzt die Klasse `App.xaml.cs` eine Liste von TaskViewModels. Für eine bessere Kapselung dieser Liste gibt es einen öffentlichen Zeiger, welcher die aktuelle Instanz des TaskViewModels repräsentiert. Bei dieser Liste können Änderungen nur durch die hierfür implementierten Methoden durchgeführt werden.

Das GuiViewModel

Im `GuiViewModel` werden die vom Server übertragenen Formulare verwaltet. Die Daten werden hier durch eine Liste von `GuiModels` repräsentiert, welche die Formulare auf die Datenstruktur abbildet.

Das Model-Package

In diesem Package werden die Models für die Datenstruktur implementiert. Um die generische Abbildung von Fragebögen zu ermöglichen, werden die Klassen in kleine Module aufgeteilt. Diese können später nahezu beliebig zusammengesetzt und verschachtelt werden können.

4.2.2 Datenstruktur

In Abbildung 4.5 ist die Datenstruktur vereinfacht dargestellt. Durch den modularen Aufbau ist eine beliebige Verschachtelung der Elemente möglich und Erweiterungen, zum Beispiel durch eine Änderung im Formularschema, einfach in die Anwendung zu integrieren. Serverseitig wird momentan eine Verschachtelungstiefe von einer Stufe unterstützt. Die Datenstruktur der Anwendung erlaubt jedoch eine beliebige Tiefe. Somit können zukünftig auch komplexere Formulare von der Anwendung verarbeitet werden. Die mit einem Stern (*) markierten Elemente sind optional, da nicht jedes Formular `CompositeFields` oder `SelectOptions` enthalten muss, jedoch mindestens ein Element vom Typ `Fields`.

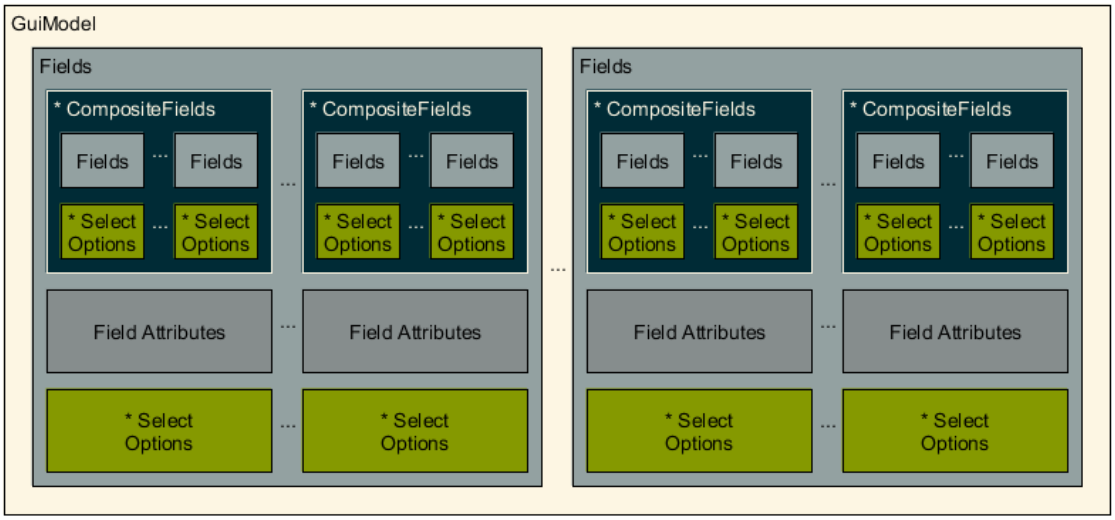

Abbildung 4.5: Datenstruktur der Anwendung

Die Klasse `SelectOptions` repräsentiert die Auswahloptionen für die in Anhang B definierten Datentypen `select, multiselect` und `radio`. Das `Fields`-Model beschreibt alle Attribute von einfachen Feldern. Falls das `Fields`-Model einen Datentyp zugewiesen bekommt, der Auswahloptionen benötigt, bekommt diese Instanz eine Liste aus `SelectOptions` zugewiesen. Wenn der Typ eines Feldes `compositeType` ist, so bekommt diese Instanz eine Liste vom Typ `CompositeFields` zugewiesen, welche wiederum eine Liste von `Fields` und `SelectOptions` enthält. Jede dieser Klassen

implementieren die Interfaces `INotifyPropertyChanged` und `IComparable`. Dadurch wird ein schneller Zugriff auf die Daten, sowie einfaches Ändern und Schreiben von Eigenschaften der Objekte gewährleistet.

4.3 Abläufe

Im Folgenden werden wichtige Prozesse erklärt, die für die weitere Implementierung und Funktionalität der Anwendung notwendig sind.

4.3.1 Serverkommunikation

Die für den Benutzer verfügbaren Aufgaben werden über die `REST-API` vom Server abgefragt. Sind Aufgaben vorhanden, werden diese in einer Auswahlliste in der Anwendung angezeigt. Wird nun eine Aufgabe ausgewählt, wird das entsprechende Formular ebenfalls über die `REST-API` vom Server angefordert. Die erhobenen Daten werden mit dem, in Anhang B.2 definierten, Antwortschema wieder an den Server zurückgeschickt.

4.3.2 Formulargenerierung

Das vom Server übertragene Formular im JSON Format wird zuerst auf seine `Version` geprüft. Wird die Version des Formulares von der Anwendung unterstützt, wird es eingelesen und in der Datenstruktur (siehe Kapitel 4.2.2) gespeichert. Um das Formular in Windows Phone anzuzeigen, muss aus diesen Daten ein XAML-String erzeugt werden. Dieser kann zur Laufzeit der Anwendung in die GUI geladen und vom Benutzer bearbeitet werden.

4.3.3 Formularvalidierung

Da die eingegebenen Daten in den Formularfeldern verschiedene Anforderungen erfüllen müssen werden diese, vor dem Zurückschicken an den Server, validiert. Das

Validierungsverfahren läuft an verschiedenen Punkten in der Anwendung ab und ist in Abbildung 4.6 schematisch dargestellt. Hierdurch kann bereits während der Eingabe eine erste Vorvalidierung vorgenommen werden.

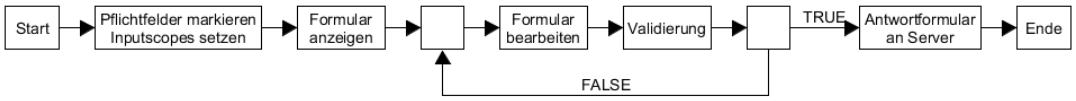

Abbildung 4.6: Ablauf der Validierung von Formularen

Zuerst werden bereits bei der Formulargenerierung Pflichtfelder für den Benutzer visuell gekennzeichnet. Außerdem werden durch das Setzen von `InputScopes` die Eingabemöglichkeiten für die einzelnen Felder entsprechend ihres Typs eingeschränkt. Nachdem der Benutzer das Formular ausgefüllt und bestätigt hat, wird überprüft, ob die vorausgesetzten Felder ausgefüllt sind und ob die Eingaben den vom Server mitgelieferten `Patterns` entsprechen. Nach der erfolgreichen Validierung, werden die Daten gespeichert und anschließend an den Server geschickt.

5

Technische Umsetzung

Im folgenden Kapitel werden die wichtigsten Technologien und Aspekte der Implementierung detailliert vorgestellt.

5.1 Verwendete Technologien

C# und XAML Bei der Entwicklung von Anwendungen für Windows Phone gibt es drei Auswahlmöglichkeiten von Programmiersprachen (*C++ und XAML*, *C# und XAML*, *JavaScript und HTML*). Da eine native Anwendung für Windows Phone entwickelt werden soll, wurde als Programmier- und Markupsprache die Kombination C# und XAML gewählt.

XamlReader Als ein Kernbestandteil der Anwendung zum Erstellen von Objektstrukturen und der Benutzeroberfläche wird die `XamlReader`-Klasse zum Erzeugen der Formulare verwendet. Sie generiert Laufzeitobjektstrukturen mit *Windows-Runtime-Objekten*, ge-

gen welche zur Laufzeit programmiert werden kann. Da diese Struktur jedoch von der Hauptobjektstruktur getrennt ist muss sie zuerst explizit verbunden werden, um auf die enthaltenen Objekte zuzugreifen [14].

LINQ to XML Der Zugriff auf durch den `XamlReader` erzeugte Objektstrukturen wird mit `LINQ to XML` (Language Integrated Query) implementiert und kann mit der Funktionalität von *XPath und XQuery* verglichen werden. Dadurch ist es möglich effizient auf Elemente der Laufzeitobjektstruktur zuzugreifen und deren Attribute abzufragen oder zu ändern [15].

Callback-Funktionen Ein weiterer Kernbestandteil der Implementierung sind *Callback-Funktionen*, mit deren Verwendung, durch den modularen Aufbau der Anwendung gestützt, eine effiziente Programmierung, Wartung und Erweiterbarkeit der Anwendung gewährleistet ist.

5.2 Implementierung

Zur besseren Veranschaulichung der Anwendungsimplementierung werden die einzelnen Punkte anhand des logischen Programmablaufs erklärt.

5.2.1 Callback-Funktionen

Callback-Funktionen werden an vielen Stellen der Anwendung eingesetzt und bestehen im Wesentlichen aus vier Komponenten.

- Datentyp<string> des Feldes (nach Anhang B)
- Objektorientierter Delegate (ähnlich dem Funktionszeiger in C und C++)
- Dictionary<Datentyp<string>, functionpointer>
- Callback-Funktionsaufruf

Listing 5.1 zeigt den grundlegenden Aufbau einer Callback-Funktion, wie sie in der Anwendung verwendet wird. Dadurch sind Erweiterung der Anwendung einfach umzu-

setzen, indem lediglich die Datenstruktur angepasst und die jeweilige Callback-Funktion implementiert wird.

```
class CallbackFunc {
  public Dictionary<string, functionPointer> dict =
    new Dictionary<string, functionPointer>();

  ///Constructor
  public CallbackFunc() {
    //fill Dictionary ...
    dict.Add("parseTextBox", parseTextBox);
    dict.Add("parseListBox", parseListBox);
  }

  //delegate for our callbackfunctions
  public delegate void functionPointer(object t);

  //this function will call the callbackfunctions
  private void parseForm(some, params) {
    // ... some code ...
    //Type of correspondending XML Elements as string
    //(e.g.: "TextBox", "PasswordBox", "ListBox")
    string elemType = xamlElement.GetType().Name;
    //get correspondending dlegate from Dictionary
    functionPointer fp = dict["parse" + elemType];
    //call the function and pass an object to it
    fp(object)
  }

  private void parseTextBox(object t) {
    // .. do some things with this object ...
  }
}
```

Listing 5.1: Grundlegender Aufbau von Callback Funktionen

5.2.2 Einlesen der Formulare

Um die Integrität der Datenstruktur zu gewährleisten müssen beim Einlesen, der vom Server übermittelten Formulare, alle Attribute und Felder (siehe Anhang B) überprüft werden. Fehlen vorausgesetzte Wertepaare so ist das übermittelte Formular ungültig und der Vorgang wird, mit einer entsprechenden Fehlermeldung an den Benutzer, abgebrochen. Falls optionale Wertepaare nicht gesetzt sind, werden hier jeweils dem Datentyp entsprechende Standardwerte eingesetzt.
Die *asynchrone* Methode `readJsonAsync` erzeugt eine temporäre Datenstruktur und übergibt zum Validieren des Formulars die einzelnen Wertepaare an die Funktion `verifyField`. Hier werden die jeweiligen Felder nach ihren Datentypen evaluiert und anschließend in die temporäre Datenstruktur geschrieben. Nach dem erfolgreichen Einlesen der Daten wird die temporäre Datenstruktur zur jeweiligen Instanz des `GuiViewModels` hinzugefügt.

5.2.3 Generische Erzeugung der GUI

Die für die Anzeige der Formulare verantwortliche Klasse `Task.xaml` besteht zunächst lediglich aus der *AppBar* zur Navigation und dem `ContentPanel`, welches als Container- und Wurzelelement für das generierte Formular dient.

Der XAML Parser

Beim Aufrufen einer Aufgabe durch den Benutzer wird in der zugehörigen Instanz der Klasse `Task.xaml.cs` das jeweilige, zum Formular gehörende, `GuiViewModel` als Datenkontext gesetzt. Anschließend muss aus diesem ViewModel die GUI erstellt werden. Da die Fragebögen generischer Natur sind, können hier keine vorgefertigten Tamplates für die einzelnen Felder vorimplementiert werden, sondern müssen zur Laufzeit erzeugt werden. Hierzu wird ein Parser eingesetzt, welcher im `Converter`-Package in den Klassen `FormGenerator` und `FieldGenerator` implementiert ist.
Abbildung 5.1 zeigt den schematischen Aufbau des Parsers mit den Grundabläufen. Der

Parser generiert aus den Rohdaten einen validen XAML String (siehe Anhang B.1.5). Dieser wird sukzessive für jedes Element in der Objektstruktur zusammengesetzt. Für jedes Formular wird die Klasse `FormGenerator` mit dem Formularobjekt aufgerufen. In dieser Klasse werden die notwendigen Namensräume gesetzt sowie der Container für das Formular erzeugt, in welchem sich später die einzelnen Felder befinden.
Für die Generierung des XAML-Strings der Feldern des Formulars wird eine neue Instanz der Klasse `FieldGenerator` erstellt. In dieser Klasse werden über eine Verteilfunktion die `field`-Objekte an die jeweilige Callback-Funktion delegiert, welche als Rückgabeparameter den XAML String generieren. Ist ein Feld vom Typ `CompositeField`, so wird innerhalb der aktuellen Instanz des `FieldGenerator` eine weitere Instanz dieser Klasse erzeugt, welche letztendlich den XAML-String für das gesamte `CompositeField` zurück gibt. Die Implementierung des XAML Parsers ist in Anhang A ausführlich dargestellt.

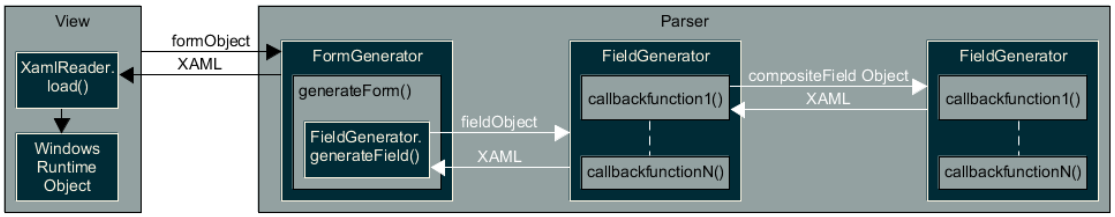

Abbildung 5.1: Schematischer Aufbau des Parsers zur Erzeugeng eines XAML Strings

Um das Formular in der Benutzeroberfläche anzuzeigen, wird mit Hilfe der `Load`-Funktion, welche durch die XamlReader-Klasse bereitgestellt wird, aus dem XAML-String die Laufzeitobjektstruktur erzeugt. Diese muss mit dem `contentPanel`-Container verknüpft werden, damit das Formular beim Laden der `View` in der Benutzeroberfläche angezeigt wird.

Abbildung 5.2 zeigt Screenshots der grafischen Benutzeroberfläche mit einem kompletten, zur Laufzeit erzeugtem Formular. Im linken Screenshot sind Textfelder und eine Textarea zu sehen. Der mittlere Screenshot zeigt eine *Multiselectbox* und ein *Composite Field* mit mehreren Teilfeldern. Ein Beispiel für eine *Singleselectbox* ist im letzten Screenshot dargestellt. Für eine bessere Übersicht für den Benutzer sind alle Elemente

gleich gestylt sowie komplexe Datenfelder (*select, multiselect, radio und composite-Field)* mit einem Rahmen abgegrenzt. Die Farbgebung richtet sich immer nach dem vom Benutzer gewählten Farbthema in den Geräteeinstellungen des Telefons.

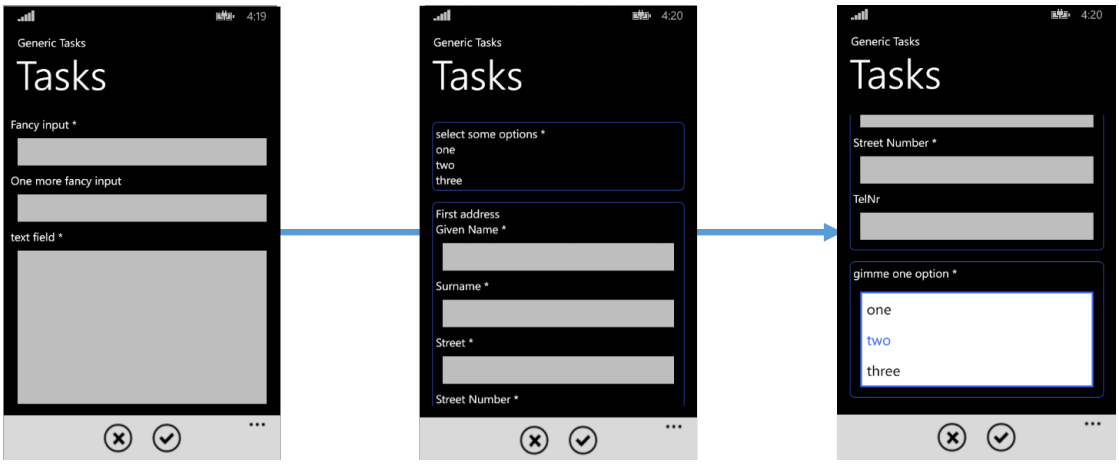

Abbildung 5.2: Anzeige des Formulars

5.2.4 Auswertung der Formulare

In der Auswertung der eingegeben Formulardaten werden diese auf das vorgegebene *Pattern* (regulärer Ausdruck) überprüft. Stimmen diese nicht überein, bekommt der Benutzer eine Rückmeldung und das entsprechende Feld im Fragebogen wird farblich markiert. Das Formular wird erst an den Server geschickt, wenn clientseitig keine Fehler mehr in der Benutzereingabe erkannt werden.
Listing 5.2 zeigt den grundlegenden Ablauf der Überprüfung von jeweiligen Objekten aus der Laufzeiobjektstruktur mittels `LINQ to XML` aufgerufen und überprüft werden. Entsprechen die Benutzereingaben nicht dem regulären Ausdruck, werden die Eigenschaften des Objekts geändert.

```
// walk through objecttree using LINQ to XML
var element = from item in ContentPanel.Children
              where item.Name = "someUniqueElementName"
              select item as FrameworkElement;

if(!VerifyElement(item)){
        notifyUser();
        // set property for this Element at Runtime
        item.Background = new SolidColorBrush(Colors.Red);
};

// check if input matches the pattern
private bool verifyElement(object item){
    // regex example: ([a-z]+\\.?)+[a-z]+
    Regex rgx = new Regex("someRegex stored in this Object");
    return rgx.IsMatch(item.Text);
}
```

Listing 5.2: Zugriff über LINQ to XML auf Objekte der Laufzeitobjektstruktur

6

Anforderungsabgleich

In diesem Kapitel werden die funktionalen und nichtfunktionalen Anforderungen aus Kapitel 3 mit der fertig gestellten Anwendung verglichen und nach ihrer Umsetzung bewertet. Die Skala läuft von 5 (voll unterstützt) bis 1 (nicht unterstützt).

6.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die Funktionsweise des Systems und wie der Benutzer damit interagieren soll. In Tabelle 6.1 sind die Anforderung mit der entsprechenden Bewertung aufgelistet und im Anschluss werden die wichtigsten Aspekte der jeweiligen Anforderung erläutert.

Nr.	Anforderung	Bewertung
FA01	Beschreibung von Formularen	5
FA02	Versionsunterstützung	5
FA03	Aufgaben vom Server laden	5
FA04	Formulare von Server laden	5
FA05	Anzeige der Formulare für die Bearbeitung	5
FA06	Benutzerauswahl der verfügbaren Aufgaben	5
FA07	Bearbeitung der Fragebögen	5
FA08	Datenerfassung der bearbeiteten Aufgaben	5
FA09	Antwortschema	3
FA10	Zurücksenden der Daten an den Server	3
FA11	Behandlung von Rückgabewerten	3
FA12	Neue beziehungsweise abhängige Aufgaben anzeigen	4
FA13	Grundlegende Benutzereinstellungen und Informationen	2

(5 voll unterstützt, 4 unterstützt, 3 teilweise unterstützt, 2 unfertig, 1 nicht unterstützt)

Tabelle 6.1: Abgleich der funktionalen Anforderungen

FA01 Beschreibung von Formularen

Die Formulare des Servers werden nach ihren vorgegebenen Spezifikationen verarbeitet. Antwortformulare werden ebenfalls nach dem definierten Schema erstellt.

FA02 Versionsunterstützung

Jedes vom Server bezogene Formular wird auf seine Version überprüft. Momentan unterstützt die Anwendung die aktuelle Version (0.0.2) des Formularschemas. Andere Versionen werden von der Anwendung abgelehnt.

FA03 Aufgaben vom Server laden

Für den Benutzer verfügbare Aufgaben können vom Server geladen und im Auswahlmenü angezeigt werden. Des Weiteren werden Beispielaufgaben angezeigt, die für den Benutzer immer zur Auswahl stehen.

FA04 Formulare vom Server laden

Bei der Auswahl einer Aufgaben wird das entsprechende Formular vom Server geladen. Außerdem stehen Beispielformulare zum Testen zur Auswahl, welche der Anwendung direkt als Ressource zur Verfügung stehen.

FA05 Anzeige der Formulare für die Bearbeitung

Vom Server bezogene Formulare im JSON können zur Laufzeit in der Benutzeroberfläche angezeigt werden und sind vollständig ausfüllbar. Damit der Benutzer eine gewohnte Arbeitsumgebung vorfindet, werden alle Formulare im gleichen Design (natives Windows Phone Design) umgesetzt.

FA06 Benutzerauswahl der verfügbaren Aufgaben

Der Benutzer kann alle angezeigten Aufgaben in beliebiger Reihenfolge auswählen. Abhängige Aufgaben werden nach Beendigung der Vorherigen automatisch angezeigt.

FA07 Bearbeitung der Fragebögen

Zur Laufzeit erzeugte Formulare sind voll funktionsfähig. Die Anwendung unterstützt alle in Anhang B definierten Datentypen.

FA08 Datenerfassung der bearbeiteten Aufgaben

Ausgefüllte Formulare können ausgewertet werden. Bei der Auswertung werden die Daten validiert. Bei Fehlern wird dem Benutzer eine entsprechende Rückmeldung gegeben.

FA09 Antwortschema

Die aus den Fragebögen erhobenen Daten können in das entsprechende Antwortschema übertragen werden. Für Daten mit dem Datentyp *Array* gibt es momentan noch Probleme.

FA10 Zurücksenden der Daten an den Server

Validierte Formulare werden an den Server geschickt. Bei einer fehlerhaften Übertragung oder bei Verbindungsfehlern wird erneut versucht das Formular an den Server zu schicken. Für Formulare mit dem Datentyp *Array* werden die Felder, falls optional ignoriert. Wenn dieses Feld vorausgesetzt wird, kann das Formular nicht zum Server geschickt werden.

FA11 Behandlung von Rückgabewerten

Rückgabewerte des Server als HTTP-Statuscode können von der Anwendung ausgewertet werden. Die auswerten der zusätzlichen Fehlerinformationen konnten nicht implementiert werden.

FA12 Neue beziehungsweise abhängige Aufgaben anzeigen

Nach dem Bearbeiten eines Formulars werden, falls verfügbar, weitere Formulare für den Benutzer zur Auswahl angezeigt, oder direkt an ein vom vorherigen Formular abhängiges Folgeformular weitergeleitet. Die Beispielaufgaben werden immer angezeigt.

FA13 Grundlegende Benutzereinstellungen und Informationen

Benutzereinstellungen und Statistiken über bereits ausgefüllte Formulare wurden nur als Dummy implementiert.

6.2 Nicht-Funktionale Anforderungen

Die nichtfunktionalen Anforderungen beschreiben die technischen Besonderheiten und Voraussetzungen der Anwendung. In Tabelle 6.2 sind die Anforderung mit der entsprechenden Bewertung aufgelistet und im Anschluss werden die wichtigsten Aspekte der jeweiligen Anforderung erläutert.

Nr.	Anforderung	Priorität
NF01	Plattformspezifisches Design	5
NF02	Toleranz gegenüber Fehleingaben	4
NF03	Toleranz gegenüber Umgebungsfehlern	5
NF04	Performanz	4
NF05	Wartbarkeit	4
NF06	Erweiterbarkeit	5
NF07	Absturzfreiheit	4
NF08	Datenschutz	1
NF09	Lokalisierung	2

(5 voll unterstützt, 4 unterstützt, 3 teilweise unterstützt, 2 unfertig, 1 nicht unterstützt))

Tabelle 6.2: Abgleich der nichtfunktionalen Anforderungen

NF01 Plattformspezifisches Design

Die Anwendung wurde vollständig auf den Windows-Phone-spezifischen Design Richtlinien aufgebaut. Besonderer Wert wurde dabei auf die Personalisierung gesetzt, in dem ausschließlich Farben aus den Benutzer-Voreinstellungen verwendet werden.

NF02 Toleranz gegenüber Fehleingaben

Durch die *verify* Funktion werden sämtliche Eingaben der Anwendung verifiziert. Außerdem wird bei kritischen Eingaben eine weitere Überprüfung durch explizite Tests durchgeführt.

NF03 Toleranz gegenüber Umgebungsfehlern

Bei Fehlern aus der Umgebung wird der aktuelle Befehl abgebrochen und der Benutzer mit einer entsprechenden Fehlermeldung informiert.

NF04 Performanz

Aufgrund der Verwendung von Nebenläufigkeit werden die meisten Aufgaben der Anwendung schnell ausgeführt. Eine Ausnahme stellt das Parsen von Formularen bis hin zur Anzeige in der Benutzeroberfläche. Je nach Komplexität eines Formulars kann dies bis zu einigen Sekunden dauern.

NF05 Wartbarkeit

Durch den starken Abstraktionsgrad der Architektur ist die Kohäsion der Module sehr hoch, ohne die Kopplung wesentlich zu erhöhen. Dies minimiert die Anzahl an Seiteneffekten und macht die Wartung einfacher.

NF06 Erweiterbarkeit

Das System bietet durch den Architekturentwurf und die Verwendung von Callback Funktionen in der Implementierung eine gute Möglichkeit für Erweiterungen.

NF07 Absturzfreiheit

Die Anwendung ist durch sorgfältiges Abfangen von Exceptions sicher gegenüber den meisten Fehler.

NF08 Datenschutz

Da die Kommunikation zur REST-API über Basic-Auth läuft, findet hier keine weitere Verschlüsselung der Daten statt. Des Weiteren ist keine Benutzerauthentifizierung beim Starten der Anwendung implementiert.

NF09 Lokalisierung

Sämtliche Texte innerhalb der Anwendung wurden mit Hilfe der AppResources Datei lokalisierbar gemacht. Es wurde jedoch keine alternative Sprache eingebaut. Texte für Aufgaben und Formulare werden vom Server übermittelt und können von der Anwendung nicht beeinflusst werden.

7

Zusammenfassung und Ausblick

Die Implementierung der Anwendung wurde in Kapitel 4 und 5 beschrieben. In Kapitel 6 wurde gezeigt, dass sämtliche in Kapitel 3 beschriebenen Anforderungen durch die Anwendung abgedeckt werden. In diesem Kapitel werden nach einer Zusammenfassung der Arbeit Kritikpunkte erläutert und anschließend ein Ausblick auf zukünftige Erweiterungen vorgestellt.

7.1 Zusammenfassung

Bei der Konzeption wurden die Anforderungen an die Anwendung analysiert und formuliert. Somit konnte der Umfang und die Funktionalität genau definiert und eine Architektur entwickelt werden, welche sich auf einem modularen Aufbau stützt. Dadurch ist es möglich, generische Formulare in einer, für Windows Phone, nativen Umgebung anzuzeigen und zu bearbeiten. Zukünftige Erweiterungen können durch die modulare Architektur

einfach in die Anwendung integriert werden. Anhand der generischen Formulare ist es möglich die Anwendung auf nahezu alle Bereiche des Mobile Crowd Sensing zu skalieren.
Alles in allem bietet die Anwendung eine native Lösung für Mobile Crowd Sensing, die durch das konsistente und plattformspezifische Design einfach zu bedienen ist.

7.2 Kritikpunkte

Während der Umsetzung ergaben sich einige Punkte, die für zukünftige Erweiterungen der Anwendung berücksichtigt werden sollten.

7.2.1 Bindings und Events

Die begrenzte Unterstützung von Data-Bindings und Events beim Erzeugen der Laufzeitobjektstruktur führt zu einer längeren Wartezeit beim Aufrufen von Formularen. Da Eventhandler nicht mit der `XamlReader` Klasse übergeben werden können, müssen diese im Anschluss zusätzlich auf die jeweiligen Objekte in der Objektstruktur registriert werden bevor das Formular bearbeitet werden kann.

7.2.2 Verfügbarkeit von Formularen

Formulare werden vom Server auf Anfrage heruntergeladen und müssen im Anschluss daran auch bearbeitet werden. Dies ist dadurch bedingt, dass die gesamte Prozesslogik eines Formulars (z.B.: Gültigkeitszeitraum) vom Server definiert wird. Somit kann die Anwendung nur mit einer aktiven Internetverbindung ausgeführt werden.

7.2.3 Lokale Speicherung

Wegen der generischen Natur der Formulare wurde auf eine persistente Zwischenspeicherung der Formulare und der eingegebenen Daten verzichtet. Diese sind nur

zur Laufzeit der Anwendung verfügbar und werden nach einem erfolgreichen Übermitteln an den Server verworfen. Daraus ergibt sich das Problem, dass wenn der Server nicht erreichbar ist und die Anwendung geschlossen wird das Formular beim nächsten Start der Anwendung erneut ausgefüllt werden muss. Dieses Problem könnte behoben werden, indem zumindest die Daten eines ausgefüllten Formulars so lange persistent gespeichert werden, bis es erfolgreich an den Server übermittelt wurde.

7.3 Ausblick

Die Anwendung kann für alle fragebogenbasierte Crowd Sensing Aufgaben, die mit den in Anhang B definierten Datentypen abgebildet werden können, verwendet werden. Um zukünftig mehr Funktionalität zu bieten und die Einsatzzwecke zu erweitern, ist es denkbar diese Datentypen durch Multimediaelemente (Bilder, Audio, Videos, etc.) zu ergänzen. Des Weiteren ist es sinnvoll eine passive Datenerfassung in die Anwendung zu integrieren und die zur Verfügung stehenden Sensoren für die Datenerfassung mit einzubeziehen, um zum Beispiel den Standort des Benutzers abzufragen.

Literaturverzeichnis

[1] Lane, N.D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., Campbell, A.T.: A survey of mobile phone sensing. Communications Magazine, IEEE **48** (2010) 140–150

[2] Ma, H., Zhao, D., Yuan, P.: Opportunities in mobile crowd sensing. Communications Magazine, IEEE **52** (2014) 29–35

[3] Guo, B., Yu, Z., Zhou, X., Zhang, D.: From participatory sensing to mobile crowd sensing. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on, IEEE (2014) 593–598

[4] Berkely, U., Nokia, NAVTEQ: Mobile Millennium. (`http://traffic.berkeley.edu/`) Besucht: 30.01.2016.

[5] Demirbas, M., Bayir, M.A., Akcora, C.G., Yilmaz, Y.S., Ferhatosmanoglu, H.: Crowd-sourced sensing and collaboration using twitter. In: World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a, IEEE (2010) 1–9

[6] Maisonneuve, N., Stevens, M., Niessen, M.E., Hanappe, P., Steels, L.: Citizen noise pollution monitoring. In: Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government, Digital Government Society of North America (2009) 96–103

[7] Pryss, R., Reichert, M., Herrmann, J., Langguth, B., Schlee, W.: Mobile Crowd Sensing in Clinical and Psychological Trials–A Case Study. In: Computer-Based Medical Systems (CBMS), 2015 IEEE 28th International Symposium on, IEEE (2015) 23–24

[8] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards process-driven mobile data collection applications: requirements, challenges, lessons learned. (2014)

[9] Masse, M.: REST API design rulebook. O'Reilly Media, Inc. (2011) ISBN: 978–1-44931-050-9.

[10] Fielding, R., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, RFC Editor (2014) `http://www.rfc-editor.org/rfc/rfc7231.txt`.

[11] Szostak, T.: Windows phone 8 application development essentials. Packt Publishing (2013) ISBN: 978-1-84969-676-0.

[12] Microsoft: Data binding for Windows Phone 8. (`https://msdn.microsoft.com/de-de/library/windows/apps/cc278072(v=vs.105).aspx`) Besucht: 13.01.2016.

[13] Bishop, J.: C# 3.0 Entwurfsmuster. Dt. Ausg., 1. Aufl edn. O'Reilly Verlag, Köln (2008) ISBN: 978-3-89721-867-3.

[14] Microsoft: XamlReader class. (`https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.markup.xamlreader.aspx`) Besucht: 10.12.2015.

[15] Microsoft: LINQ to XML Overwiev. (`https://msdn.microsoft.com/en-us/library/bb387061.aspx`) Besucht: 14.02.2016.

[16] Schindler, A.: Konzeption und Entwicklung einer modularen, ereignisgesteuerten Server-Client-Architektur zur Crowd-basierten Datenerfassung mit mobilen Endgeräten (2015) Masterarbeit an der Universität Ulm.

A

Quelltexte

A.1 Die FormGenerator Klasse

Im Folgenden wird die Implementierung des XAML Parsers vorgestellt. Listing A.1 zeigt die `FormGenerator`-Klasse, welche beim Aufruf eines Formulars das jeweilige `GuiModel` übergeben bekommt. Damit das Formular zur Laufzeit generiert werden kann, müssen zuerst die Container für die Formularfelder mit den jeweiligen Namespaces erzeugt werden. Für die Generierung der Felder wird eine Instanz der `FieldGenerator`-Klasse erzeugt.

```
1  class FormGenerator {
2    internal static string generateForm(Model.GuiModel guiModel) {
3      StringBuilder formString = new StringBuilder();
4      // startString for XAMLReader: set required namespaces for he XamlReader
5      String startForm = "<ScrollViewer xmlns=\"http://schemas.microsoft.com
6      /winfx/2006/xaml/presentation\" xmlns:x=\"http://schemas.microsoft.com
7      /winfx/2006/xaml\" xmlns:toolkit=\"clr-namespace:Microsoft.Phone.Controls;
8      assembly=Microsoft.Phone.Controls.Toolkit\" x:Name=\"Content\"
9      VerticalScrollBarVisibility=\"Auto\">" +
10     "<StackPanel x:Name=\"stackContent\" Orientation=\"Vertical\"
11     Visibility=\"Visible\">";
```

```
12      formString.Append(startForm);
13
14      //init new Field Generator
15      FieldGenerator fgen = new FieldGenerator();
16      //Create XAML string for each Field
17      foreach (Model.Fields f in guiModel.Fields) {
18        var _inputType = f.InputType;
19        formString.Append(fgen.generateField(_inputType, f));
20      }
21
22      // endString for XAML
23      String endForm = "</StackPanel>" +
24        "</ScrollViewer>";
25      formString.Append(endForm);
26      // return complete XAML to XamReader
27      return formString.ToString();
28    }
29  }
```

Listing A.1: FormGenerator.cs

A.2 Die FieldGenerator Klasse

Listing A.2 zeigt die Implementierung der `FielGenerator`-Klasse. Zur besseren Übersicht sind hier nur ausgewählte Funktionen der Klasse dargestellt. Für die anderen Datentypen erfolgt die Implementierung analog. Eine Ausnahme bildet hier der Datentyp `CompositeField`, bei welchem innerhalb der `Fieldgenerator`-Klasse eine neue Instanz dieser Klasse erzeugt wird.

```
1  class FieldGenerator {
2
3    public delegate string functionPointer(Model.Fields field);
4    public Dictionary<string, functionPointer> dict =
5      new Dictionary<string, functionPointer>();
6
7    public FieldGenerator(){
```

```
8     dict.Add("createtext", createText);
9     dict.Add("createtextarea", createText);
10    dict.Add("createemail", createText);
11    dict.Add("createtel", createText);
12    dict.Add("createurl", createText);
13    dict.Add("createmultiselect", createMultiselect);
14    dict.Add("createselect", createSelect);
15    // build composite field
16    dict.Add("createcompositeField", createCompositeField);
17  }
18
19  /// <summary>
20  /// Main processing function to delegate to correct callbackfunction
21  /// </summary>
22  /// <param name="_inputType"></param>
23  /// <param name="field"></param>
24  /// <returns></returns>
25  public string generateField(string _inputType, Model.Fields field){
26    functionPointer fp = dict["create" + _inputType];
27    return fp(field);
28  }
29
30  /// <summary>
31  /// creates label for fields if none is set but required there
32  /// will be a star "*"
33  /// </summary>
34  /// <param name="field"></param>
35  /// <returns></returns>
36  private string createLabel(Model.Fields field) {
37    if (!field.Required && (field.Label.Equals("") || field.Label == null)) {
38          return "";
39    } else {
40      if(field.InputType.Equals("compositeField"))
41        return "<TextBlock Text=\"" + field.Label + "\"/>";
42      string req = field.Required ? "*" : "";
43      return "<TextBlock Text=\"" + field.Label + " " + req + "\"/>";
44    }
45  }
```

```
46
47    /// <summary>
48    /// build XAMl for Textboxes
49    /// </summary>
50    /// <param name="field"></param>
51    /// <returns></returns>
52    private string createText(Model.Fields field) {
53      string text = "";
54      // add Label
55      text = text + createLabel(field);
56      // add TextBox
57      //is there a max length for input set?
58      string maxLength = field.MaxLength==0 ? "" : " MaxLength=\"" +
59        field.MaxLength + "\"";
60      //placeholdertext set?
61      string placehold = field.PlaceholderText.Equals("") ? "" : " Text=\""
62        + field.PlaceholderText + "\"";
63      // is it a textarea?
64      string area = field.InputType.Equals("textarea") ? " Height=\"300\"" : "";
65      // set InputScopes if necessary
66      string numberinput = field.InputType.Equals("number") ?
67        " InputScope=\"Number\"" : "";
68      string emailinput = field.InputType.Equals("email") ?
69        " InputScope=\"EmailUserName\"" : "";
70      string telinput = field.InputType.Equals("tel") ?
71        " InputScope=\"TelephoneNumber\"" : "";
72      string urlinput = field.InputType.Equals("url") ?
73        "InputScope=\"Url\"" : "";
74
75      //build final textbox
76      text = text + "<TextBox x:Name=\"" + field.Name +
77        "\" TextWrapping=\"Wrap\"" +
78        placehold +
79        maxLength +
80        area +
81        numberinput +
82        emailinput +
83        telinput +
```

```
84        urlinput + "/>";
85      // return String to FormGenerator
86      return text;
87    }
88
89    /// <summary>
90    /// builds mutliselect form
91    /// </summary>
92    /// <param name="field"></param>
93    /// <returns></returns>
94    private string createMultiselect(Model.Fields field) {
95      StringBuilder formString = new StringBuilder();
96      string border = "<Border BorderThickness=\"1,1,1,1\"
97        CornerRadius=\"8,8,8,8\" Padding=\"5\" Margin=\"10\"
98          BorderBrush=\"{StaticResource PhoneAccentBrush}\">" +
99        "<StackPanel x:Name=\"" + field.Name + "bordeer\">";
100     formString.Append(border);
101     formString.Append(createLabel(field));
102     string startform = "<ListBox x:Name=\"" + field.Name + "\"
103       SelectionMode=\"Multiple\">";
104     formString.Append(startform);
105     // add select options
106     foreach (var obj in field.SelectOptions) {
107       string label = obj.OptionsLabel;
108       formString.Append("<ListBoxItem Content=\"" + label + "\"/>");
109     }
110
111     string endform ="</ListBox>";
112     formString.Append(endform);
113     string endborder = "</StackPanel></Border>";
114     formString.Append(endborder);
115     // return String to FormGenerator
116     return formString.ToString();
117   }
118 }
```

Listing A.2: FieldGenerator.cs

B

Formulare

Ein Kernbestandteil der Anwendung sind Formulare. Sie beschreiben den Aufbau von Fragebögen und dienen zur Kommunikation und Ergebnisübermittlung zum Server. Der Aufbau der Formulare ist in [16] definiert und wird im Folgenden zusammenfassend dargestellt und auf die plattformspezifische Anwendung mit Windows Phone eingegangen.

B.1 Input Schema

Die Formulare sind über das JSON Schema definiert und bestehen aus vier Abschnitten. Das Input Schema beschreibt die Formulare, wie der Client sie vom Server ausgeliefert bekommt.

B.1.1 Versionen

Im ersten Abschnitt (`version`) sind die Basisinformationen über die Versionsnummer des Protokolls enthalten. Die Versionsnummer dient zur Identifizierung des Schemas. Somit kann auch die von der mobilen Anwendung unterstützten Versionen verifiziert werden.

B.1.2 Header

Das zweite Feld (`header`) bietet Platz für zusätzliche Metainformationen zum Formular. Diese sind beispielsweise die Transaktions-ID (`transactionID`) und die Lebensdauer (`transactionExpires`). Listing B.1 zeigt die Response des Servers mit Transaktions-ID und Lebensdauer in Sekunden.

```
1 { "form":{
2         "header":{
3         "transactionId": "h7erpH98zer",
4         "transactionTTL": 1800,
5         "transactionExpires": "2015-07-28T14:32:48Z" }}}
```

Listing B.1: Antwortformular mit Transaktions-ID und Lebensdauer [16]

B.1.3 Fields

Formularfelder werden im dritten Abschnitt, dem `fields` Array beschrieben. Hierbei haben die Felder je nach Typ verschiedene Attribute und Datentypen, welche für die Anzeige und die Eingabevalidierung erforderlich sind. Im Folgenden werden die Eigenschaften nach [16] zusammengefasst und ihre Verwendung im Kontext zu Windows Phone dargestellt.

Attribute

- `name` (required, unique) Beschreibt den eindeutigen Namen des Formularfeldes und wird für die Datenübertragung verwendet. Des Weiteren beschreibt der Name das Feld in der Datenstruktur der Anwendung und den eindeutigen Namen im *XAML* String des Elements.
- `label` (optional) Repräsentiert die Beschriftung des Formularfeldes in der GUI. Bei einem *compositeField* wird es als Überschrift eines Fomrularblocks verwendet.

In der Beschreibung der grafischen Oberfläche mit *XAML* wird dieses Feld durch einen Textblock (`<TextBlock />`) dargestellt.

- `type` (required) Gibt den Typ des Datenfeldes an und besteht weitestgehend aus einer Untermenge der HTML5 *Input-Types*[1]. Für die mobile Anwendung werden die jeweiligen Input-Types auf *UI controls für Windows Phone*[2] angepasst und für eine bessere Benutzereingabe die entsrechenden `InputScopes` gesetzt.
- `compositeField` (conditional) In diesem Feld wird ein Formularblock referenziert, welcher im JSON im `compositeFields` Array definiert ist. Das Feld wird nur beachtet, wenn im Feld `type` der Typ `compositeField` gesetzt ist.
- `pattern` (optional) Übergibt einen regulären Ausdruck für die Einschränkung und Validierung der Benutzereingaben.
- `required` (optional) Dieses Feld enthält die Werte `true` oder `false` und gibt an, ob das Feld vom Benutzer ausgefüllt werden muss. Beim Fehlen dieses Attributs wird als Standardwert `true` gesetzt.

Datentypen

Da die bei Eingabefeldern bei Windows Phone meist die gleichen *controls* verwendet werden und sich nur einzelne Attribute im XAML String ändern, sind die Datentypen nach [16] im Folgenden zu ihren jeweiligen *controls* für Windows Phone zusammengefasst.

- `compositeField` Markiert ein `compositeField`, welches eine bestimmte Gruppe von Feldern im `compositeFields` Array referenziert.
- `text, textarea, number, email, tel, url` Bei diesen Datentypen handelt es sich im Allgemeinen um Eingabefelder für beliebigen Text. Sie unterscheiden sich wesentlich durch ihre Attribute. Im `XAML`-String sind diese Felder alle durch eine `<TextBox ... />` repräsentiert. Für die jeweiligen Datentypen wird der zugehörige `InputScope` gesetzt, welcher zum Beispiel aus einem Textfeld ein Eingabefeld für Zahlen macht.

[1] https://www.w3.org/TR/html5/forms.html#attr-input-type
[2] https://msdn.microsoft.com/de-de/library/windows/apps/ff402561(v=vs.105).aspx

- `password` Hier werden Eingaben als Punkte dargestellt, aber auch im Klartext übergeben (`XAML` repräsentation: `<PasswordBox .../>`).
- `date` Dient zur Eingabe eines Datums. Hierbei kann das Format yyyy-mm-dd oder mm-dd sein. Bei Windows Phone wird hierzu die Kontrollstruktur `<DatePicker ... />` verwendet.
- `time` Bietet Eingabemöglichkeiten für Uhrzeiten, welche durch einen `<TimePicker ... />` repräsentiert werden.
- `datetime` Stellt eine Komposition aus `time` und `date` dar.
- `select` Bietet die Auswahl einer Option aus mehreren vorgegebenen Möglichkeiten an. Für diese Kontrollstruktur wird eine Erweiterung der `ListBox` verwendet, welche die Anzeige als Dropdown ermöglicht.
- `multiselect` Bei diesem Datentyp können bis zu n Optionen aus n Möglichkeiten ausgewählt werden. Die Mehrfachauswahl wird duch das `SelectionMode="Multiple"` Attribut in einer `ListBox` ermöglicht.
- `checkbox` Dienen zum Ein- und Ausschalten von Optionen durch Freilassen oder Auswählen der Kontrollstruktur (`<CheckBox ... />`).
- `radio` Erlaubt die Auswahl einer Option aus einer Liste von mehreren Optionen. Dieser Typ bietet im Gegensatz zu `select` eine flache Auflistung, welche auch in Gruppen zusammengefasst werden können.

B.1.4 Composite Fields

Composite Fields sind gruppierte Felder, welche in einem Feld als selbst definierter Datentyp referenziert werden. Dadurch ist es möglich solche Gruppierungen mehrmals zu verwenden und sie mit Individuellen Überschriften zu versehen. Ein Beispiel dafür ist das Adressfeld, welches für eine Rechnungsadresse und für eine Lieferadresse verwendet werden kann. Durch die serverseitige Definition [16] ist momentan eine maximale Verschachtelungstiefe von einer Stufe angedacht. Dies bedeutet, dass ein

`compositeField` keine weiteren Felder vom selben Typ enthalten darf, sondern nur noch Felder mit einfachen Datentypen.

B.1.5 Beipiel für Input Schema

Listing B.2 zeigt ein Beispiel eines Formularschemas, wie die Anwendung es vom Server übertragen bekommt. In Listing B.3 ist der entsprechende XAML-Code dargestellt, welcher von der Anwendung zur Laufzeit generiert wird.

```
{ "form":{
    "version":"0.0.2",
    "fields":[
      { "name":"fancyInput",
        "label":"Fancy input",
        "type":"text",
        "pattern":"([a-z]+\\.?)+[a-z]+",
        "required":true },
      { "label":"text field",
        "name":"fancyText",
        "type":"textarea" },
      { "name":"someOption",
        "label":"select some options",
        "type":"multiselect",
        "options":[
          { "label":"one",
            "value":"o1" },
          { "label":"two",
            "value":"o2" },
          { "label":"three",
            "value":"o3" }]},
      { "name":"fancyAddress",
        "type":"compositeField",
        "compositeField":"address",
        "label":"First address" },
      { "name":"somesingleOption",
        "label":"gimme one option",
        "type":"select",
        "options":[
```

```
          { "label":"one",
            "value":"o1" },
          { "label":"two",
            "value":"o2" },
          { "label":"three",
            "value":"o3" }]}],
    "compositeFields":[
      { "name":"address",
        "fields":[
          { "name":"givenName",
            "label":"Given Name",
            "type":"text",
            "maxlength":100,
            "required":true },
          { "name":"surname",
            "label":"Surname",
            "type":"text",
            "maxlength":100,
            "required":true },
          { "name":"street",
            "label":"Street",
            "type":"text",
            "maxlength":100,
            "required":true },
          { "name":"streetNumber",
            "label":"Street Number",
            "type":"text",
            "pattern":"[0-9]+(/[0-9]*[a-z]?)?",
            "required":true },
          { "name":"tel",
            "label":"TelNr",
            "type":"tel",
            "required":false }
          ]
      }]
  }}
```

Listing B.2: Gesamtbeispiel JSON Formular

Beim in Listing B.3 dargestellten XAML-Code wurden zur besseren Übersicht die Definitionen der Namensräume und Style Attribute (z.B. bei `<Border ... />` Elementen) ausgelassen. Das Verhalten des `required` Attributes wird bei der Formularauswertung durch die Implementierung abgedeckt. Felder, für die `"required": true` gesetzt ist, sind im zugehörigen Label mit einem Stern (*) markiert.

```
<ScrollViewer ... >
  <StackPanel ... >
    <TextBlock Text="Fancy input *" />
    <TextBox x:Name="fancyInput" TextWrapping="Wrap" />
    <TextBlock Text="text field *" />
    <TextBox x:Name="fancyText" TextWrapping="Wrap" Height="300" />
    <Border ... >
      <StackPanel x:Name="someOptionbordeer">
        <TextBlock Text="select some options *" />
        <ListBox x:Name="someOption" SelectionMode="Multiple">
          <ListBoxItem Content="one" />
          <ListBoxItem Content="two" />
          <ListBoxItem Content="three" />
        </ListBox>
      </StackPanel>
    </Border>
    <Border ... >
      <StackPanel x:Name="fancyAddress">
        <TextBlock Text="First address" />
        <TextBlock Text="Given Name *" />
        <TextBox x:Name="givenName" TextWrapping="Wrap" MaxLength="100" />
        <TextBlock Text="Surname *" />
        <TextBox x:Name="surname" TextWrapping="Wrap" MaxLength="100" />
        <TextBlock Text="Street *" />
        <TextBox x:Name="street" TextWrapping="Wrap" MaxLength="100" />
        <TextBlock Text="Street Number *" />
        <TextBox x:Name="streetNumber" TextWrapping="Wrap" />
        <TextBlock Text="TelNr " />
        <TextBox x:Name="tel" ... InputScope="TelephoneNumber" />
      </StackPanel>
    </Border>
    <Border ... >
```

```
33        <StackPanel x:Name="somesingleOptionborder">
34          <TextBlock Text="gimme one option *" />
35          <toolkit:ListPicker x:Name="somesingleOption">
36            <toolkit:ListPickerItem x:Name="o1" Content="one" />
37            <toolkit:ListPickerItem x:Name="o2" Content="two" />
38            <toolkit:ListPickerItem x:Name="o3" Content="three" />
39          </toolkit:ListPicker>
40        </StackPanel>
41      </Border>
42    </StackPanel>
43  </ScrollViewer>
```

Listing B.3: XAML Darstellung des Beispiels aus Listing B.2

B.2 Antwortschema

Das Antwortschema nach [16] für die ausgefüllten Formulare wird als JSON an den Server geschickt. In Listing B.4 ist ein Antwort-JSON dargestellt, welches die wichtigsten Elemente aus Listing B.2 mit ihren definierten Antworttypen darstellt. Dabei werden als Antwort für einfache Elemente `Strings` und für Mehrfachauswahlen ein `Array` mit den ausgewählten Elementen zurückgegeben. Bei `compositeField` Elementen wird ein `Array` zurückgegeben, welches wiederum aus den Antworten wie bei oben genannten Elementen aufgebaut ist.

```
1  { "formreply": {
2    "version": "0.0.2",
3    "fields": [
4    { "name": "fancyInput",
5      "value": "i wrote some fancy stuff" },
6    { "name": "fancyInputAgain",
7      "value": "i wrote even more fancy stuff, because i can" },
8    { "name": "someOption", "value": ["o1, o2] },
9    { "name": "fancyText",
10     "value": "so text\nmuch wow!\ncan even write multiple lines here" },
11   { "name": "fancyAddress",
12     "value": [
```

```
13      { "name": "givenName", "value":" Sherlock" },
14      { "name": "surname", "value": "Holmes" },
15      { "name": "street", "value": "Baker Street" },
16      { "name": "streetNumber", "value": "221b" } ]
17    }]
18 }}
```

Listing B.4: Antwortschema eines Formulars [16]

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Georg Heinz Eisenhart Matrikelnummer: 797131

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Georg Heinz Eisenhart